Computed Items

You can create new data items in the OLAPQuery, Results, Pivot, Results and Chart report sections by building equations to compute data items, or by applying functions to existing data items. Computed items are like normal data items, and can be included in reports or re-used to compute other data.

For example, you can modify the Amount Sold item by building an equation around it, multiplying by the Unit Price item and renaming the resulting item 'Revenue'. You can also apply a scalar function such as Cume to Amount Sold and return each individual value as a cumulative running total, or simply multiply Amount Sold by the local tax rate to find the tax owed on each sale.

Tip In the Query section, you can modify data items as they are retrieved from the server, or add new computed items. In Results and Pivot, you can add and modify them.

Important In BrioQuery 6.0, Brio moved from its own proprietary scripting language to the industry-standard, JavaScript. BrioQuery uses a JavaScript engine that does not accept European or non-US number formats. See the Brio JavaScript Reference on-line help for more information about using JavaScript.

To compute or modify a data item

1. On the shortcut menu, click **Add Computed Item** to add a new item, or select an item and click **Modify Column** to modify an existing computed item.

2. Rename the item to reflect the result of the computation.

3. Define the new data item by building an equation in the definition box.

· Click Functions to apply scalar functions to Reference items using the Functions dialog box.

· Click Reference to display the Reference dialog box, and select Request line items to place in the equation, or use as function arguments.

· Use the buttons provided in the Computed Item dialog box to insert arithmetic and logical operators into the equation at the insertion point.

· You can also type item, function and operator names directly in the panel. The names are case-sensitive, but you will need to replace spaces in item names with underscores ('_').

4. When the equation is complete, click **OK**.

Results computed items are displayed in the Outliner in blue.

Field Definitions

Available fields and options for this dialog box are:

Name Assign the name of the column title that you want to appear in the computed column. The default title name for each computed item added to the report is "Computed", then "COMPUTED2" and so on (sequentially numbered). If you assign a name to a COMPUTED ITEM, which is identical to an existing scalar function name, BrioQuery numbers the title name starting with number 2. For example, if you name a column Rank, which already exists as a scalar function name, BrioQuery reads and displays the name Rank2.

Definition Build a computation by adding items from the Variables, Functions and Operators panels, or by typing directly into the panel at the cursor. The maximum length of a computed item definition is limited to the text length of 32 K.

Operator buttons Click the buttons to add arithmetic or logical operators to a computation in the Definition panel. Operators are added at the insertion point.

For Operator definitions, see the Operator table below.

Functions Apply a function to a selected data item in the Definition panel.

Reference Display the Reference dialog box to add items from the Request line to the equation in the Definition panel.

Tip You can apply multiple scalar functions to selected items within the equation.

Options Display the Data Type list box and set the data type when performing mixed-data type computations.

For more information about this feature, see Adjusting Data Types.

Operator buttons

Click the following buttons to add arithmetic logical operators, mixed expressions, and apply functions to a computation in the Definition panel. Operators are added at the insertion point.

Arithmetic operators An arithmetic operators take numerical values (either logical or variables) as their operands and return a single numerical value.

+ Add

- Subtract

Tip! If a computed item appears on a Query's Request Line, and that item's definition uses subtraction, such as "Mytable.Column1-5", a SQL error can occur. The exact error depends on the database, but the most common error would indicate an undefined name was used. Because databases allow hyphenated names, BrioQuery attempts to deal with such names intuitively. Thus, an item definition like "Mytable.Column1-5" will be interpreted as a name. In order to ensure it is treated as subtraction, include a space on either side of the hyphen/subtraction operator. For example, entering the computed item definition as Mytable.Column1 - 5" will ensure that correct SQL is generated.

* Multiply

/ Divide

(Begin sub-operations

) End sub-operations

++ Increment

-- Decrement

In the Query section, the following Operator buttons can be used only at the local metatopic level.

\+ Add

\- Subtract

* Multiply

/ Divide

(Begin sub-operations

) End sub-operations

In the Results section, all Operator buttons are available.

Arithmetic An arithmetic operators take numerical values (either logical or

Operator	variables) as their operands and return a single numerical value. In the following example, var1 has been assigned the value 5 and var2 has been assigned the value 4.
Mod (%)	The modulus operator returns the remainder of dividing var1 by var2. For example, 5 % 4 returns 1.
Comparison Operators	A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical ordering. In the following example, var1 has been assigned the value 3 and var2 has been assigned the value 4.
= =	Returns true if the operands are equal. For example, 3 == var1
!=	Returns true if the operands are not equal. For example, var1!=4
<	Returns true if left operand is less than right operand. For example, var1 < var2
<=	Returns true if left operand is less than or equal to right operand. For example, var1 < = var2 var2 < = 5
>	Returns true if left operand is greater than or equal to the right operand. For example, var2 > var1
>=	Returns true if left operand is greater than right operand. For example, var2 > = var1 var1 > = 3
Statements	Executes a set of statements if a specified condition is true. If the condition is false, another set of statements can be executed.
if…else	if executes a set of statements if a specified condition is true. The specified condition may be another statement and can include other nested if statements. Braces, {}, must enclose multiple statements. If the condition is false, another set of statements can be executed if the optional else statement has been included in the script. A sample if … else statement looks likes this:

```
if (condition) {
statements1
}
else {
statements2
}
```

Logical Operators Logical operators take Boolean (logical) values as operands and return a Boolean value.

AND (&&) Use the logical operator AND to connect two conditional expressions and retrieve records only if each expression is true, since this determines if a condition is true.

Computed items will not be retrieved if any condition belonging to a conditional expression is false.

The AND logical operator is usually nested within another conditional expression, for example, expressions which use if and else statements.
For example
if ((OS == ‘Windows’) && (Item_Type == ‘Modem’)) {‘Windows’} else {‘other’}

OR (||) Use the or logical operator to specify a combination of expressions and retrieve records that include at least one of the expressions. For example, if one of the words is Washington or Oregon, every record with the expression "Washington" and every record with the word "Oregon" is included.

Typically the OR logical operator is nested within other conditional expression, for example, expressions which use if and else logical operators. For example if you want to assign Washington and Oregon to the "Northwestern Region" and all other states to "Other Regions", you write:

if ((State = = ‘Washington’)|| (State == ‘Oregon’)) {‘Northwestern Region’} else {‘Other Regions’}

NOT (!) Use the NOT logical operator to compute and show items more accurately stated in a negative way. In effect, all records are retrieved except those that fulfill the conditional expression.

You enter the conditional expression with the NOT (!) logical operator preceding the conditional expression.:

The conditional expression can be a simple value or nested within other conditional expressions, for example, expressions using AND and OR. A combined condition expression that uses NOT is true if the conditional expression following NOT is false. A combined conditional expression is false if the conditional expression following NOT is true. For example, suppose you are looking to list all states that are not in the Northwestern region. In this case, you enter the conditional expression:

if (! (State = = 'Northwestern Region')) {‘Other Regions’}

Tips

- Type the word null (no quotes) into the expression panel to represent null values.
- All text string constant values entered in expressions must be enclosed in single quotes. All date constant values entered in expressions must be enclosed in quotes. Numbers can be entered without quotes.
- To join items with a space or other character, simply reference or type items and strings into the expression panel and join them with the + operator (for example, City + ‘, ‘ + State). To join without additional characters, use the Concat function.
- In division operations, the divisor may not be null or equal to zero. If a data item serves as the divisor in an expression (for example, 5000 / Units_Sold) and includes null or zero values, first create a computed item using if/else logic to remove null and zero values, and then compute the item containing the division operation.
- Two date items may be subtracted, but not added. The Add Month function adds an integer value to a date.
- You may not nest functions inside the Sum, Cume, Chr, and Breaksum functions.

For a detailed description of JavaScript operators, see the online Brio JavaScript Reference.